REMARKS

The Office Action of September 30, 2005 has been carefully considered. In response thereto, the claims have been amended as set forth above.

Claims 1-5 were rejected as being anticipated by the O'Conner. Claims 1 and 4 have been amended to more clearly distinguish over the cited reference. Reconsideration is respectfully requested.

The present invention relates, in one aspect, to a computational method in which one functional unit, during execution of an instruction, outputs data to or receives input data from another functional unit *during execution of the instruction*, where the functional units share a common memory, e.g., a micro code memory. Claims 1, 4 and 6 have been amended to make clear that such input/output occurs during execution of the instruction. Figures 6a and 6b illustrate a particular computation (that of Figure 5, which in turn references the 2Dtransform operation of Figure 2) performed conventionally (Figure 6a) and using the computational method of the present invention (Figure 6b). In the illustrated example, the computational method of the invention results in a 20% reduction in computation time.

The foregoing example is described more particularly in the present specification at page 10, line 27, to page 6, line 19. In Figure 6a, the 2Dtransform operation (Figure 2) is performed *atomically*; i.e., inputs are presented simultaneously to a complex functional unit, which performs the operation and presents outputs simultaneously. In Figure 6b, by contrast, inputs are not presented simultaneously, nor are outputs presented simultaneously. The input i_1 (= p + q) is presented to the complex functional unit as soon as it is available, in cycle 1. During cycles 1 and 2, the second input i_2 (= p – q – 2) is prepared and is then presented to the complex functional unit in cycle 3. Also in cycle 3, the first output o_1 (= $2*i_2$ + 3) is presented. During cycles 3, 4 and 5, the output o_2 (= $5*i_1$ + $2*i_2$ + 1) is computed and presented. During cycles 4 and 5, o_1 (previously made available in cycle 3) is squared and the value 100 is subtracted to form a partial result as

part of the condition checking in the last statement of Figure 5. In cycles 6 and 7, o_1 is squared and the resulting quantity added to the partial result. Finally, in cycle 8, the inequality is evaluated. In this example, it may be seen that output data (e.g., o_1) of the complex functional unit ("first functional unit") is processed by the other functional unit ("second functional unit") during execution of the instruction (e.g., 2Dtransform) by the complex functional unit ("first functional unit") as recited in claim 1. Also, input data (e.g., i_2) to the first functional unit is generated by the second functional unit during execution of the instruction as recited in claim 1.

During a prior amendment, the claims had been amended to recite that one functional unit, during execution of an instruction, outputs data to or receives input data from another functional unit *in the midst of execution of the instruction*. This language was believed to be more express than the original language. Based on the Examiner's interpretation, however, the amended language introduced ambiguity into the claim. Accordingly, the claims have been amended to revert to the original language.

The O'Connor reference teaches something quite different. O'Connor relates to "scoreboarding," a method of resource arbitration in which a centralized set of tables ("registers") in the CPU keep track of what is being used and when. Each row indicates what is being used at each clock tick. Scoreboarding is used to manage dispatch, stalling, and completion of instructions, to watches for Write-After-Write, Read-After-Write and Write-After-Read (WAW, RAW, WAR) hazards, and to manage the execution sequence (e.g. possible instruction reordering) to avoid these hazards. Hazards are detected by observing register references and operation types.

In particular, O'Connor describes a hardware-efficient implementation of a bypass circuit that that enables a result from one execution unit to be provided to a waiting execution unit at the same time that result is sent to a register.

In O'Connor, one execution unit waits for results from another execution unit in order for the first execution unit to begin execution. The execution units do not operate concurrently to achieve execution of an instruction.

Claims 1-5 were rejected as being anticipated by McNeill. This rejection is respectfully traversed.

The rejection ignores the basic distinction between an *instruction* and a computing *task*. An instruction is a well-defined operation that takes a known number of clock cycles of a functional unit. A computing task, on the other hand, is performed by executing a series of instructions in sequence. The time required to perform the computing task is often indeterminate.

In McNeill, a slave CPU performs a search on data supplied through a disk interface. Such a search is a computing task. It is not "an instruction" as set forth in the present claims.

Accordingly, claims 1 and 4 are believed to patentably define over the cited references.

Dependent claims 2, 3, and 5 are also believed to add novel and patentable subject matter to their respective independent claims. Withdrawal of the rejection and allowance of claims 1-5 is respectfully requested.

Respectfully submitted,

Michael J. Ure, Reg. 33,089

Dated: ~~September 1~~ Dec. 27, 2005